Generating MIDI music

Aleš Calábek, 469489

spring 2021

1 Introduction

The goal of this project is to generate classical music with an LSTM architecture of neural networks using symbolic representation of music in the MIDI format.

The most common way to store music is by encoding the audio waveform tracking the change of the amplitude in time. To properly capture all details of the sound the sampling rate has to be sufficiently high, usually around 48 kHz, i.e. 48,000 samples per second. This approach results in relatively large file size since much of the information is redundant – sound is periodic oscillation and there can be thousands of identical cycles before a significant change.

On the other hand, symbolic representation describes music on a more abstract level, only storing qualitative information such as pitch, duration, volume, timbre, etc. Besides the smaller file size, a great advantage of the symbolic representation is that we can work with it in the form of text and apply common NLP techniques.

Nevertheless, generating good music is a very hard problem comparable to generating poetry or other long and cohesive texts. The main difficulty lies in the development of a musical motif, phrasing and the general structure over the whole piece of music [1].

In this project, I compare my results with two articles [2][3] that implement a solution using a Generative Adversarial Network (GAN) and a Variational Autoencoder (VAE).

2 Theory

In order to use the LSTM model we first need to encode the MIDI files into a suitable text representation. I used the *py-midicsv* tool to convert the MIDI into a CSV format of which you can see an example in Figure 1. For this project the most important rows are the ones that contain the *Note_on_c* event indicating that a note has been played. The other attributes on this row specify the Track number, Time, Channel, Pitch and Volume of the note.

To further optimize the data size I have discarded the irrelevant control sequences and extracted only valuable data into a custom ABC notation. ABC notation is a simplified form of writing down music mainly used for simple

```
1, 1920, Tempo, 472404
2, 1920, Note_on_c, 0, 65, 0
2, 1920, Note_on_c, 0, 64, 37
4, 1935, Control_c, 0, 64, 0
4, 1998, Control_c, 0, 64, 127
2, 2040, Note_on_c, 0, 64, 0
2, 2040, Note_on_c, 0, 55, 34
2, 2160, Note_on_c, 0, 55, 0
2, 2160, Note_on_c, 0, 60, 34
2, 2280, Note_on_c, 0, 60, 0
2, 2280, Note_on_c, 0, 64, 32
```

Figure 1: Every row in the CSV file contain at least three fields – Track number, Time, Type of event – and optionally further fields depending on the type of the event.

children's songs. However, it is not suitable for complete description of a piece of classical music. I have adopted the basic idea of a note followed by its length and a delimiter indicating progress in time. Furthermore, every note is aligned to a multiple of a 32nd note which I have chosen as the base unit of time. For an example of this notation see Figure 2.

As the final encoding step, I one-hot encoded the text and split it into training sequences.

The model I am using in this project is a Long-short term memory (LSTM) neural network. The advantage over standard Multilayer Perceptron is its ability to work with sequences, and compared to a simple Recurrent Neural Network it can take longer history into consideration.

The method of generating music with this model is to first approach it as a classification problem – predicting the next character based on a sequence of preceding characters. The LSTM internally learns a distribution on the characters and we can then randomly sample from this distribution to generate new text. What follows is a simple conversion from text back to MIDI.

```
β>γ β0θ β>Ϗ β0γ βSγ βVγ βYγ βVγ βSγ βVγ βSγ β0γ βSγ βJγ βMγ βLγ βJγ β0ω<ωLωOωTω ŸHε δJε δLε δMη ζOβMβLδ δQε δJδ δOη ζQγ β0γ
βMγ βLγ βMγ βLγOε βJγ βHγQε βJγ βHγSε βGγ βEεTη δNε βVβTβOνSδ δXε δQδ δNγVη βLγ βNδ βXγ βJεVγ βTγ βOεSβ βOγ βCεMεQγ βSγ
βEεLεTγ βSγ βGεJεTγ βVγ βHεHηXγ βVγ βXγ βJβZγHβGδ[ε δLεOιSε δEδTδ δJηMγQε βLγ βMλVγ βLγTγ βJγSγ βHγQγ βGδOη βMγ β<εHιLε βOγ
β>εJιMγ βLγ β@εFεMγ βOγ βAηEδHεQγ βOγ βJεMεQγ βCβSγAβ@δCδTρ βOγ βEεHεMγ βLγ β>δMδ βEγ βCηGγJε βHγ βJιOρSι βEγ βCγ βAγ β@βCι
βAγ β@γTε β>γ β<βVε β>γ β<γCεXε β;γ β9δEεYη δ>εGε β[βYβEεHηXδ δBε]ε βJβHβCγGδVδ βEγ βFγLε[η βCγ β=εEδ β]γ β>εJη[γ βYγ βEιXδ
βLγ βJγ]η βHγ β@ιGε β_γ βLν]γ β[γ βEγYρ βGγ βHγOε βJγ βHγJιQε βGγ βEγSε βCγ βHεTηXη βGγ βHγ βEβVβZγTβLγSδ[ρ βJγ βHγXε βGγ
βHβQδ βEγ βGγVη βHγ βJγZι βHγXγ βGγVγ βEγTγ βCιSε[γ βYγ β7εXγ βVγ β9εTγ βVγ β;εJεTγ βSγ β<ηLεQγ βTγ βNεSγ β>βQγ<β;δOη δ@ε
βQβOβ9δNδ βTγ β>ηSε βQγ βLδPδ β@γ β>γQηXε β<γ β;εVε βSγ β@εQγTγ βOγSγ βBεNεQγ βPγ β>εDεMεQγ βSγ β<εEηLεTβ βNγ βAιJηQγ
βGβPγEβDδSε βLγ β@ιHεNγQγ βPγSγ βBδQγTδ βPγ β>ιGηQγYε βSγ βPγXε βHγNγ β@εGγPγVι βEγQγ βAεDδSε δ@γEιTγ β>γSγ β@ιSαTβSζ γDε
βQγ β9ιEιQι δHε δJε δCεLε δEεMη δGε βOβMβHηLδ δQε βJβHβGδJδ δ7εLεOη δ9εEδ βQγ β;εJιOγ βMγ β<ηLδ δCεLεOε β>β<β;δJιNεQε
δ@εOθSε δ9δHδTη δ>ηEεNε βVβTβLιPεSδ β@γ β>γQιTε β<γ β;δJεVε δ:εOεXε δ9εQεYη δ7εSε β[βYβ9δTηXδ δBεEε]ε βVβTβCεGεSδVδ
δ@εIεXε[η δ>θJηQδ β]γ βVη[γ βLβYγJβ@εHδXδ βXγ βAεMεVγ]ε βTγ βCηGδSδVδ δLη[ε^ε βEβCβAδUε]ε βMγ βFεLγVε[γ βJγYγ β@δIεXδ[γ βYγ
βEηUε[γ βXγ βVεYγ βFγ[γ βEγXε[α]β[αCγYβ[βAγQε]β β@γUγ βAβEεVγ β>γ[γ βCεGεXαYβXζ γEεIεLε βVγ β>σJηNεVε δOε βLβJβHδQη δNε
βSβQβGδOδ β@γ β>γLηTε β<γ β;γNβ β9γNγ β7γLγSη β6γJγ β4δHσ βTγ β@εSγ βQγ βBεOγ βNγ βCιLγOε βJγ βLιQε βJγ βEγHγSε βCγGγ
βBδEγJλTη βCγ β>εEβ βBγVβTβCδCθSδ βQγ βGεOγXε βMγ βHεLγQδ βOγ βJεMγVη βQγ βLηOλ βXγ βVγ βMβTγLβJδSγ βQγTγ βOεRεVγ βXγ
```

Figure 2: Example of my custom ABC notation. Each note represented as an ASCII character is followed by its length encoded as a letter of the Greek alphabet for better readability. A space character indicates progress in time.

3 Implementation

3.1 Installation

The project is written in Python 3.8.5 and all necessary dependencies can be installed with the *pip* package installer using the command:

```
$ pip3 install -r requirements.txt
```

Additionally, the structure of this project as well as usage of the main functions can be found in the *README.md* file.

3.2 Data preprocessing

I have collected over 10 hours of classical music by various composers from [4] and some more data from [5] were used for evaluation.

The *make_corpus.py* script tries to convert all MIDI files into the notation described in the previous section. Unfortunately, in many cases the *py-midicsv* tool is unable to properly read the file due to corruption and different MIDI versions.

Furthermore, I check each piece of music for its key and transpose it into a key with no sharps or flats (C major or A minor). Since the distribution of notes in one key is similar, the models should be able to learn the distribution more easily.

3.3 Training the models

I have implemented three models based on the LSTM architecture using the Keras and tensorflow libraries:

- *simple_lstm* consists of only one LSTM and one Dense layers and serves as a baseline,
- *lstm* consists of more LSTM and more Dense layers,
- *cnn_lstm* tries to use convolutional layers in combination with LSTM.

The full architecture of each model can be found in the *models.py* file.

One problem I encountered was the large size of the one-hot encoded training dataset which would require hundreds of gigabytes of RAM. For that reason I had to iteratively train the models on individual batches of data instead of giving them the entire dataset.

The models were trained for about one day each on a powerful desktop computer and I used the Early Stopping mechanism to prevent them from overfitting.

3.4 Generating music

To begin the generation process the model needs a seed – a sequence of characters as an initial input. I have tried using an empty sequence made of space characters, but found that the model is unable to produce diverse enough results. Therefore, I decided to use a random sequence from the training corpus instead.

I also found that squaring and normalizing the probability distribution before sampling the next character helps the models achieve better harmony. Without this modification the result seemed quite out of tune.

Another peculiar behavior of the models is the high tempo of the generated music. I haven't been able to explain this anomaly but it is trivial to fix this problem in post-production and achieve the intended result.

Overall, the models produce unusable results most of the time and interesting sequences have to be cherry-picked. However, the ratio of acceptable to poor results is significantly better for the more complex models compared to the baseline.

4 Evaluation

Since generating music stems from a classification problem, categorical cross-entropy was used as an appropriate loss metric. Additionally, expected log likelihood of a time step and accuracy have been used as is common in other articles on this topic. Table 1 shows values of these metrics as evaluated on a test corpus and we can see that the baseline model achieved best scores across all metrics. However, it later becomes obvious from the qualitative evaluation that these results have little to no relevance to the quality of the generated music itself.

	loss ↓	LL ↑	acc ↑
simple LSTM	2.113	-12.529	0.466
LSTM	2.674	-13.708	0.466
CNN-LSTM	2.951	-13.159	0.412

Table 1: Surprisingly, the simplest model shows best performance according to all metrics.

In order to assess the quality of the music I have conducted a survey. I first generated 20 samples with each model and picked two 10 second long passages. This procedure also matches that of the articles with which I compare results. I then collected the opinions of 8 respondents that rated the samples using five criteria – rhythm, melody, harmony, coherence and overall score – on a scale from 1 to 5 (5 being the best). To get a better idea of what score we are aiming for, I included a sample composed by a real human composer in the survey. Amusingly, even the human composition has not reached the full score. Nevertheless, it still beats all machine learning models.

Table 2 shows the mean score for all models in each of the five criteria. Here we can clearly see that the simple baseline model lags behind the other models. On the other hand, the LSTM seems to outperform the CNN-LSTM architecture, and is comparable to the GAN and VAE. We can also see that all models are good with rhythm and struggle a bit more with melody and coherence.

	Rhythm	Melody	Harmony	Coherence	Overall
simple LSTM	2.94	1.94	2.56	2.63	2.19
LSTM	4.25	3.25	3.31	3.75	3.38
CNN-LSTM	3.67	2.5	3.00	3.19	2.75
MuseGAN[2]	3.25	2.81	2.92	3.00	2.93
Sandwich2[3]	4.26	3.68	4.08	3.22	3.62
Human	4.38	4.63	4.00	4.38	4.38

Table 2: The LSTM's performance is comparable to that of the MuseGAN and Sandwich2 VAE models. Still, they don't seem to reach the quality of a human composer.

5 Conclusion

I have implemented three models based on the LSTM architecture that can generate new classical music in the MIDI format. Most of the times the results are poor but interesting passages can be cherry-picked as inspiration for a human composer. I have evaluated the models using several metrics and conducted a user survey to assess the quality of the generated music. I also compared them to results of other articles.

As a future extension, the models could be trained on even longer sequences, hopefully capturing more distant dependencies in the music. Other genres of music could be experimented with as well, although the lack of training data in the MIDI format might be an obstacle. Finally, architectures based on the attention mechanism that are lately gaining on popularity could be investigated.

References

[1] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. "Deep learning techniques for music generation–a survey". In: *arXiv preprint arXiv:1709.01620* (2017).

[2] Hao-Wen Dong et al. "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[3] Xia Liang, Junmin Wu, and Jing Cao. "MIDI-Sandwich2: RNN-based Hierarchical Multi-modal Fusion Generation VAE networks for multi-track symbolic music generation". In: *arXiv preprint arXiv:1909.03522* (2019).

[4] URL: https://www.kaggle.com/soumikrakshit/classical-music-midi.

[5] URL: https://www.kaggle.com/blanderbuss/midi-classic-music.